Safe Controller Optimization for Quadrotors with Gaussian Processes

Felix Berkenkamp, Angela P. Schoellig, and Andreas Krause

***Abstract*— One of the most fundamental problems when designing controllers for dynamic systems is the tuning of the controller parameters. Typically, a model of the system is used to obtain an initial controller, but ultimately the controller parameters must be tuned manually on the real system to achieve the best performance. To avoid this manual tuning step, methods from machine learning, such as Bayesian optimization, have been used. However, as these methods evaluate different controller parameters on the real system, safety-critical system failures may happen. In this paper, we overcome this problem by applying, for the first time, a recently developed safe optimization algorithm, SafeOpt, to the problem of automatic controller parameter tuning. Given an initial, low-performance controller, SafeOpt automatically optimizes the parameters of a control law while guaranteeing safety. It models the underlying performance measure as a Gaussian process and only explores new controller parameters whose performance lies above a safe performance threshold with high probability. Experimental results on a quadrotor vehicle indicate that the proposed method enables fast, automatic, and safe optimization of controller parameters without human intervention.**

SUPPLEMENTARY MATERIAL

A video demonstrating the proposed safe, automatic controller optimization on a quadrotor vehicle can be found at http://tiny.cc/icra16_video. A Python implementation of the algorithm is available in [1].

I. INTRODUCTION

Tuning controller parameters is a challenging task, which requires significant domain knowledge and can be very time consuming. Classical approaches to automate this process, such as the ones in [2] and [3], either rely on model assumptions (e.g., linearity), which may be the very reason why the initial, model-based controller performs poorly, or require gradient approximations, which are difficult to obtain from noisy measurements. Methods without these assumptions, such as genetic algorithms [4], typically require an impractical number of evaluations on the real system. Moreover, all these methods may converge to a local optimum.

In this paper, we present a method to automatically tune controller parameters without requiring a model of the underlying, dynamic system or the computation of gradients. Additionally, our approach guarantees safety during the convergence to the global optimum and only requires few experiments (see Fig. 1).

Felix Berkenkamp and Andreas Krause are with the Learning & Adaptive Systems Group (LAS), Department of Computer Science, ETH Zurich, Switzerland. Email: {befelix, krausea}@ethz.ch

Angela P. Schoellig is with the University of Toronto Institute for Aerospace Studies (UTIAS), Canada. Email: schoellig@utias.utoronto.ca

This research was supported in part by SNSF grant 200020_159557, NSERC grant RGPIN-2014-04634, and the Connaught New Researcher Award.

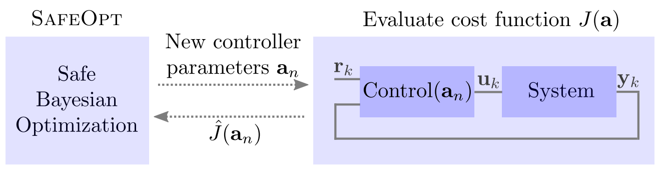

SafeOpt
Safe Bayesian Optimization
New controller parameters $\mathbf{a}_n$
$\hat{J}(\mathbf{a}_n)$
Evaluate cost function $J(\mathbf{a})$
$\mathbf{r}_k$
Control($\mathbf{a}_n$)
$\mathbf{u}_k$
System
$\mathbf{y}_k$

Fig. 1. Overview of the algorithm. The safe Bayesian optimization algorithm selects new, safe parameters at which the performance function is evaluated on the real system. Based on the noisy information gained from the experiment, the algorithm chooses a new, informative and safe evaluation point at each iteration n. This is repeated until the optimum is found.

In general, the goal of automatic controller tuning is to find controller parameters through experiments that optimize a given performance measure. Yet the function that maps controller parameters to performance values is unknown *a priori*. Finding the global optimum of an unknown function is an impossible task. However, by making assumptions about the regularity of the unknown function, the field of Bayesian optimization has developed practical optimization algorithms (cf. [5]) that provably find the global optimum, while evaluating the function at only a few parameter combinations [6], [7]. Another major advantage of these methods is that they explicitly model noise in the performance function evaluations. Bayesian optimization methods often model the unknown function as a Gaussian process (GP) [8], which can guide function evaluations to locations that are informative about the optimum of the unknown function [5], [9].

Bayesian optimization has been used in robotics to automate the process of tuning controller parameters. Examples include gait optimization of legged robots [10], [11] and controller optimization for a snake-like robot [12]. These papers show that Bayesian optimization reliably finds the optimal controller parameters within a few experiments. In [13] the controller parameters of a state-feedback controller were automatically tuned using Bayesian optimization. In this work, the cost matrices in the LQR framework were used as a low-dimensional representation of parameters, which made the method applicable to higher-dimensional systems. A comparison of different Bayesian and non-Bayesian global optimization methods can be found in [10].

Despite the experimental success of Bayesian optimization methods, they have one weakness in real-world experiments. While gradient-ascent methods, such as [2], typically improve the controller at every iteration and thereby ensure that the resulting controllers continue to be stable, informative samples in Bayesian optimization are typically far away from the original control law to gain maximum information. This often leads to the evaluation of unstable controllers and system failures early on in the optimization process.

In this paper, we overcome the problem of safety by using a modified version of SafeOpt [14], a recently developed safe Bayesian optimization algorithm that builds on the results from [7] and, in addition, guarantees safety by only evaluating parameters that achieve a safe performance threshold with high probability. The algorithm retains the desirable properties of normal Bayesian optimization, such as the ability to find the safely reachable optimum [14]. The result is a safe and automatic controller tuning algorithm, which is illustrated in Fig. 1.

Other approaches that guarantee system safety in the presence of unmodeled dynamics make assumptions about and explicitly model the uncertainties in the nominal model. In this setting, the controllers are gradually improved by estimating the unmodeled dynamics from experimental data and recomputing the control law based on this estimate. Stability can be guaranteed by ensuring that either the controller is robustly stable for all possible models within the uncertainty specification [15] or the system never leaves a safe subset of the state space [16], [17], [18]. Both methods require a system model and uncertainty specification to be known *a priori*, which must be accurate enough to guarantee stability. In contrast, the method presented in this paper only requires the controller structure and initial, safe controller parameters to be known. We do not use a model of the system but model the performance function directly.

The problem of Bayesian optimization subject to constraints on the performance function was previously studied in [19]. However, they did not consider this constraint as safety-critical. As a result, evaluating parameters that do not satisfy the constraints was allowed, which would mean, in our case, evaluating unsafe parameters. In contrast, we explicitly avoid the evaluation of unsafe controller parameters. The problem of safe exploration, without the goal of optimizing the performance, was considered in [20]. In this paper, we explore and also optimize the function within the safe region.

We demonstrate the safety and performance of SafeOpt experimentally on a quadrotor vehicle, for which we automatically learn optimal controller parameters without failures during the experiments. Early results of the approach were presented in [21].

II. PROBLEM STATEMENT

This section introduces the safe optimization problem considered in this paper. We assume that we have a nonlinear, dynamic control law of the form

$$\mathbf{u}_k = \mathbf{g}(\mathbf{y}_k, \mathbf{r}_k, \mathbf{a}_n), \tag{1}$$

which may include internal states (e.g., integrators). The control law is parameterized by controller parameters, $\mathbf{a}_n \in \mathcal{A}$, at iteration n in a domain $\mathcal{A}$. At time step k, the controller maps the noisy measurements of a dynamic system, $\mathbf{y}_k$, and a reference signal, $\mathbf{r}_k$, to control actions, $\mathbf{u}_k$. The controller aims to achieve a desired objective, such as reference tracking. This control objective is specified in terms of a performance measure, $J(\mathbf{a})\colon \mathcal{A} \mapsto \mathbb{R}$, which is evaluated on the real system and assigns higher values to controllers with better performance. The performance measure can be anything in general, but typically depends on the control inputs and output errors of the closed-loop system (see Fig. 1). It is evaluated over a finite time horizon. For example, in [10] the average walking speed of a bipedal robot over three experiments was used.

The goal is to automatically find the controller parameters, $\mathbf{a}$, that maximize the performance measure, $J(\mathbf{a})$, based on noisy evaluations of J for different controller parameters, $\hat{J}(\mathbf{a}_n) = J(\mathbf{a}_n) + \omega$, where $\omega \sim \mathcal{N}(0, \sigma_\omega^2)$ is zero-mean Gaussian noise. We assume that the system is safety-critical; that is, the optimization algorithm must ensure safety when evaluating new controller parameters. We do not assume any knowledge about the model of the dynamic system, which means that the dependence of J on $\mathbf{a}$ is unknown *a priori* and needs to be learned as part of the optimization routine. Additionally, the optimization procedure must be sample-efficient; that is, only few evaluations of the performance function should be carried out in order to save time and avoid system wear. To start the optimization procedure safely, we assume that an initial set of stabilizing controller parameters (with potentially poor performance) is available.

In this paper, we encode the safety criterion as a performance threshold, $J_{\min}$, below which we do not want to fall; that is, $J(a_n) \geq J_{\min}$ must hold with high probability for all $\mathbf{a}_n$ at which J is evaluated. With this definition of safety, the resulting controllers are likely to be stable, since unstable systems typically have a significantly lower performance when considering a sufficiently long time horizon.

III. METHODOLOGY

In this section, we review GPs and Bayesian Optimization and illustrate the theory behind SafeOpt.

A. Gaussian Process (GP)

The function $J(\mathbf{a})$ in Sec. II is unknown *a priori*. We use a nonparametric model to approximate the unknown function over its domain $\mathcal{A}$. In particular, we use a GP to approximate $J(\mathbf{a})$.

GPs are a popular choice for nonparametric regression in machine learning, where the goal is to find an approximation of a nonlinear map, $J(\mathbf{a})\colon \mathcal{A} \mapsto \mathbb{R}$, from an input vector $\mathbf{a} \in \mathcal{A}$ to the function value $J(\mathbf{a})$. This is accomplished by assuming that function values $J(\mathbf{a})$, associated with different values of $\mathbf{a}$, are random variables and that any finite number of these random variables have a joint Gaussian distribution depending on the values of $\mathbf{a}$ [8].

For the nonparametric regression, we define a prior mean function and a covariance function, $k(\mathbf{a}_i, \mathbf{a}_j)$, which defines the covariance of any two function values, $J(\mathbf{a}_i)$ and $J(\mathbf{a}_j)$, $i, j \in \mathbb{N}$. The latter is also known as the kernel. In this work, the mean is assumed to be zero without loss of generality. The choice of kernel function is problem-dependent and encodes assumptions about smoothness and rate of change of the unknown function. A review of different kernels can be found in [8]. More information about the kernel used in this paper can be found in Sec. V.

The GP framework can be used to predict the function value, $J(\mathbf{a}^*)$, at an arbitrary input, $\mathbf{a}^* \in \mathcal{A}$, based on a set of n past observations, $\mathcal{D}_n = \{\mathbf{a}_i, \hat{J}(\mathbf{a}_i)\}_{i=1}^n$. We assume that observations are noisy measurements of the true function value, $J(\mathbf{a})$; that is, $\hat{J}(\mathbf{a}) = J(\mathbf{a}) + \omega$ with $\omega \sim \mathcal{N}(0, \sigma_\omega^2)$. Conditioned on the previous observations, the mean and variance of the prediction are given by

$$\mu_n(\mathbf{a}^*) = \mathbf{k}_n(\mathbf{a}^*)(\mathbf{K}_n + \mathbf{I}_n\sigma_\omega^2)^{-1}\hat{\mathbf{J}}_n, \quad (2)$$

$$\sigma_n^2(\mathbf{a}^*) = k(\mathbf{a}^*, \mathbf{a}^*) - \mathbf{k}_n(\mathbf{a}^*)(\mathbf{K}_n + \mathbf{I}_n\sigma_\omega^2)^{-1}\mathbf{k}_n^{\mathrm{T}}(\mathbf{a}^*), \quad (3)$$

where $\hat{\mathbf{J}}_n = \left[\hat{J}(\mathbf{a}_1), \dots, \hat{J}(\mathbf{a}_n)\right]^{\mathrm{T}}$ is the vector of observed, noisy function values, the covariance matrix $\mathbf{K}_n \in \mathbb{R}^{n \times n}$ has entries $[\mathbf{K}_n]_{(i,j)} = k(\mathbf{a}_i, \mathbf{a}_j)$, $i, j \in \{1, \dots, n\}$, and the vector $\mathbf{k}_n(\mathbf{a}^*) = \left[k(\mathbf{a}^*, \mathbf{a}_1), \dots, k(\mathbf{a}^*, \mathbf{a}_n)\right]$ contains the covariances between the new input $\mathbf{a}^*$ and the observed data points in $\mathcal{D}_n$. The identity matrix is denoted by $\mathbf{I}_n \in \mathbb{R}^{n \times n}$.

B. Bayesian Optimization

Bayesian optimization aims to find the global maximum of an unknown function [5]. The assumption is that evaluating the function is expensive, while computational resources are cheap. This fits our problem in Sec. II, where each evaluation of the performance function corresponds to an experiment on the real system, which takes time and causes wear.

In general, Bayesian optimization models the objective function as a random function and uses this model to determine informative sample locations. A popular approach is to model the underlying function as a GP, see Sec. III-A. GP-based methods use the mean and variance predictions in (2) and (3) to compute the next sample location. For example, [7] evaluates, at iteration n, the parameters

$$\mathbf{a}_n = \operatorname*{argmax}_{\mathbf{a} \in \mathcal{A}} \mu_{n-1}(\mathbf{a}) + \beta_n \sigma_{n-1}(\mathbf{a}), \quad (4)$$

where β_n is a iteration-varying scalar that defines the confidence interval of the GP. Intuitively, (4) selects new evaluation points at locations where the upper bound of the confidence interval of the GP estimate is maximal. Repeatedly evaluating the system at locations given by (4) improves the mean estimate of the underlying function and decreases the uncertainty at candidate locations for the maximum, such that the global maximum is found after a finite number of iterations, cf. [7].

While (4) is also an optimization problem, solving it does not require any evaluations on the real system but only uses the GP model. This corresponds to the assumption of cheap computational resources.

C. Safe Bayesian Optimization

In this paper, we build upon the safe optimization algorithm SafeOpt [14]. SafeOpt is a Bayesian optimization algorithm, see Sec. III-B, which aims to maximize an unknown function by modeling it as a GP over a finite set of parameters $\mathcal{A}$. However, instead of optimizing the underlying function globally, it restricts itself to a safe set $\mathcal{S} = \{\mathbf{a} \in \mathcal{A} \mid J(\mathbf{a}) \geq J_{\min}\}$, which only contains parameters that lead to a performance value above the safe threshold, $J_{\min}$. This safe set is not known initially, but is estimated after each function evaluation. In our case, the initial, safe set, $\mathcal{S}_0$, corresponds to the initial, safe controller parameters, $\mathbf{a}_0$.

In this setting, the challenge is to find an evaluation strategy similar to (4), which at each iteration n not only aims to find the global maximum within the currently known safe set $\mathcal{S}_n$ (exploitation), but also to increase the set $\mathcal{S}_n$ of controllers that are known to be safe (exploration). SafeOpt provides a solution to this problem [14] by choosing for the next experiment the safe controller parameters about whose performance we are most uncertain. Parameters are chosen from two sets: the set of potential maximizers, $\mathcal{M}_n$, whose values can lie above the current maximum according to the GP estimate, and the set of potential expanders, $\mathcal{G}_n$, which can expand the set of safe controllers, $\mathcal{S}_n$ (see Fig. 2).

In [14] these two sets were estimated using a Lipschitz constant. However, in practical applications this represents an additional tuning parameter. In the next section, we modify the algorithm in [14] to use the GP's prediction directly in order to estimate these sets.

IV. MODIFIED SafeOpt ALGORITHM

In this section, we modify SafeOpt from [14], to work without the specification of a Lipschitz constant. Additionally, we provide an implementation in Python [1], which is significantly faster than a naive implementation. In contrast to the original algorithm in [14], we estimate the sets, $\mathcal{S}_n, \mathcal{G}_n$, and $\mathcal{M}_n$, directly from the GP. In particular, we define the upper and lower bound of the confidence interval at iteration n as

$$u_n(\mathbf{a}) = \mu_{n-1}(\mathbf{a}) + \beta_n \sigma_{n-1}(\mathbf{a}), \quad (5)$$

$$l_n(\mathbf{a}) = \mu_{n-1}(\mathbf{a}) - \beta_n \sigma_{n-1}(\mathbf{a}), \quad (6)$$

where $\beta_n \in \mathbb{R}_+$ defines the confidence interval that we want to achieve. For example, in our experiments we use $\beta_n = 2$. Using these bounds, we define the safe set as all the parameters $\mathbf{a}$ that are very likely to lead to function values above the safe performance threshold, $J_{\min}$, according to the GP estimate,

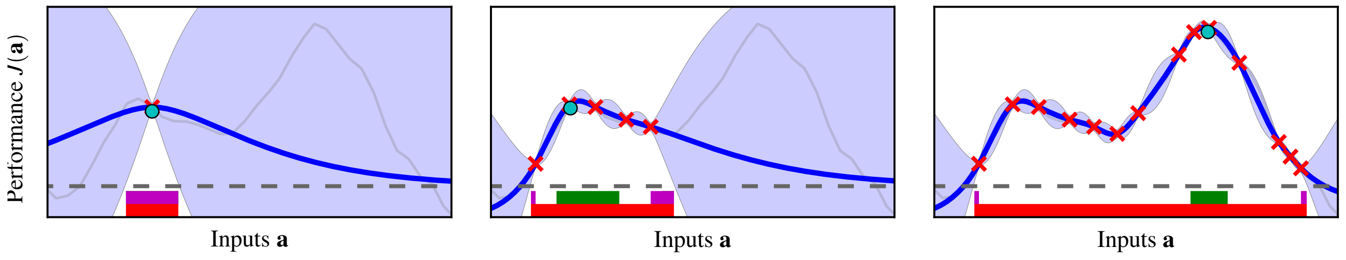
$$\mathcal{S}_n = \{\mathbf{a} \in \mathcal{A} \mid l_n(\mathbf{a}) \geq J_{\min}\}. \quad (7)$$

The set of potential maximizers contains all safe parameters that could obtain the maximum value given the high-probability bounds in (5) and (6). It is given by the set of safe parameters for which the upper confidence interval, u_n, is above the best, safe lower bound:

$$\mathcal{M}_n = \left\{\mathbf{a} \in \mathcal{S}_n \mid u_n(\mathbf{a}) \geq \max_{\mathbf{a}' \in \mathcal{A}} l_n(\mathbf{a}')\right\}. \quad (8)$$

The set of potential expanders is more difficult to define without the Lipschitz constant, as it quantifies whether new parameters could be classified as safe after a new measurement. We define an optimistic indicator function for expanders,

$$g_n(\mathbf{a}) = \left|\left\{\mathbf{a}' \in \mathcal{A} \setminus \mathcal{S}_n \mid l_{n,(\mathbf{a}, u_n(\mathbf{a}))}(\mathbf{a}') \geq J_{\min}\right\}\right|, \quad (9)$$



(a) Initial, safe parameters. (b) After 5 evaluations: local maximum found. (c) After 13 evaluations: global maximum found.

Fig. 2. Optimization with modified SafeOpt algorithm after 1, 5 and 13 evaluations of the performance measure. Based on the mean estimate (blue) and confidence interval (light blue), the algorithm selects safe evaluation points above the safe threshold $J_{\min}$ (black dashed) from the safe set $\mathcal{S}_n$ (red), which are either potential maximizers $\mathcal{M}_n$ (green) or expanders $\mathcal{G}_n$ (magenta). It then learns about the function by drawing noisy samples from the unknown, underlying function (light gray). This way we expand the safe region (red) as much as possible, and, simultaneously, find the global optimum of the unknown function (13) (cyan ball).

where $l_{n,(\mathbf{a},u_n(\mathbf{a}))}$ is the lower bound of the GP, (6), based on past data and an artificial data point $(\mathbf{a}, u_n(\mathbf{a}))$ with a noiseless measurement of the upper confidence bound. The function in (9) counts how many previously unsafe points can be classified as safe according to (7) assuming that we measure $u_n(\mathbf{a})$ when evaluating $J(\mathbf{a})$. This function is positive if the new data point has a non-negligible chance to expand the safe set. Consequently, the set of possible expanders is defined as

$$\mathcal{G}_n = \{\mathbf{a} \in \mathcal{S}_n \mid g_n(\mathbf{a}) > 0\}. \tag{10}$$

For a graphical representation of the three sets, see Fig. 2.

As in [14], we choose new parameters at which to evaluate the performance on the real system by selecting the parameters about which we are the most uncertain from the union of the sets $\mathcal{G}_n$ and $\mathcal{M}_n$; that is, at iteration n we choose to evaluate the function at $\mathbf{a}_n$,

$$\mathbf{a}_n = \underset{a \in \mathcal{G}_n \cup \mathcal{M}_n}{\operatorname{argmax}} \; w_n(\mathbf{a}), \tag{11}$$

$$w_n(\mathbf{a}) = u_n(\mathbf{a}) - l_n(\mathbf{a}). \tag{12}$$

This evaluation criterion has many desirable properties, including the ability to find the safely reachable optimum [14]. In particular, it works well for expanding the safe set [20], while at the same time trading-off exploration and exploitation. For the exploration, the most uncertain parameter locations are usually on the boundary of the safe set, which results in efficient exploration. Typical kernel functions can only classify states in the vicinity of past observations as safe, which leads to a coarse sampling of the safe parameter space. The coarse samples already provide information about the maximizers in $\mathcal{S}_n$ during exploration. For example, all points from the set $\mathcal{M}_5$ in Fig. 2b are eliminated as potential maximizers in Fig. 2c, as we observe larger values during the safe exploration. We obtain an estimate of the best currently known parameters from

$$\underset{\mathbf{a} \in \mathcal{S}_n}{\operatorname{argmax}} \; l_n(\mathbf{a}), \tag{13}$$

which corresponds to the point that achieves the best lower bound on the performance.

A summary of the entire algorithm is found in Algorithm 1. It starts by computing the sets $\mathcal{S}_n, \mathcal{G}_n$ and $\mathcal{M}_n$ in Lines 3–5. Afterwards, a new evaluation point is chosen from the sets $\mathcal{M}_n$ and $\mathcal{G}_n$ in Line 6, and the real system is evaluated in Line 7. Finally, the GP is updated with the new, noisy measurement in Line 8. This process is repeated until either the algorithm is aborted by the user or until a desired confidence, defined by $\max_{\mathbf{a} \in \mathcal{G}_n \cup \mathcal{M}_n} w_n(\mathbf{a})$, is reached [14].

Algorithm 1: Modified SafeOpt algorithm

```
  Inputs: Domain $\mathcal{A}$
          Safe threshold $J_{\min}$
          GP prior $(k(\mathbf{a}_i, \mathbf{a}_j),\ \sigma_\omega^2)$
          Initial, safe controller parameters $\mathbf{a}_0$
1 Initialize GP with $(\mathbf{a}_0,\ \hat{J}(\mathbf{a}_0))$
2 for $n = 1, \dots$ do
3     $\mathcal{S}_n \leftarrow \{\mathbf{a} \in \mathcal{A} \mid l_n \geq J_{\min}\}$
4     $\mathcal{M}_n \leftarrow \{\mathbf{a} \in \mathcal{S}_n \mid u_n(\mathbf{a}) \geq \max_{\mathbf{a}'} l_n(\mathbf{a}')\}$
5     $\mathcal{G}_n \leftarrow \{\mathbf{a} \in \mathcal{S}_n \mid g_n(\mathbf{a}) > 0\}$
6     $\mathbf{a}_n \leftarrow \operatorname{argmax}_{a \in \mathcal{G}_n \cup \mathcal{M}_n} w_n(\mathbf{a})$
7     Obtain measurement $\hat{J}(\mathbf{a}_n) \leftarrow J(\mathbf{a}_n) + \omega_n$
8     Update GP with $(\mathbf{a}_n,\ \hat{J}(\mathbf{a}_n))$
9 end
```

Computing the complete set $\mathcal{G}_n$ in (10) is computationally expensive, since we have to recompute the matrix inverse in (2) and (3) for every point in $\mathcal{S}_n$. However, since Algorithm 1 only selects the most uncertain parameter in Line 6, it suffices to find the expander in $\mathcal{S}_n \setminus \mathcal{M}_n$ with the largest value w_n above the maximum variance in $\mathcal{M}_n$, $\max_{\mathbf{a} \in \mathcal{M}_n} w_n$. As a result, it suffices to iterate over the points in $\{\mathbf{a} \in \mathcal{S}_n \setminus \mathcal{M}_n \mid w_n(\mathbf{a}) > \max_{\mathbf{a}' \in \mathcal{M}_n} w_n(\mathbf{a}')\}$ in order of decreasing values w_n and stop the computation as soon as an expander is found. This significantly reduces computation time, since typically only few or no parameters need to be checked as expanders using (9).

It is possible to extend this algorithm to additional constraints that do not depend on the performance, such as constraints on inputs or states. Please refer to [22] for details.

V. Quadrotor Experiments

In this section, we demonstrate the algorithm on a quadrotor vehicle, a Parrot AR.Drone 2.0. A video of the experiments can be found at http://tiny.cc/icra16_video. The quadrotor learns optimal controller gains for the position controller in x-direction. The other two directions and the heading angle are stabilized by separate controllers. The system's dynamics can be described by four states: position, x, velocity, $\dot{x}$, pitch, ϕ, and angular velocity, ω. Measurements of all states are available from an overhead motion capture camera system. The control input, u, is the desired pitch angle, which in turn is the input to an unknown, proprietary, on-board controller. We define a linear control law, which computes the control input at time k:

$$u_k = k_1(x_k - r_k) + k_2\dot{x}_k. \tag{14}$$

The control law depends on the reference position r_k and is parameterized by two parameters, $\mathbf{a} = (k_1, k_2)$.

The goal is to find controller parameters that maximize the performance during a 1-meter reference position change. For an experiment with parameters $\mathbf{a}_n$ at iteration n,

$$J(\mathbf{a}_n) = C(\mathbf{a}_n) - 0.95C(\mathbf{a}_0), \tag{15}$$

$$C(\mathbf{a}_n) = -\sum_{k=0}^{N} \mathbf{x}_k^{\mathrm{T}}\mathbf{Q}\mathbf{x}_k + Ru_k^2, \tag{16}$$

where, to compute the cost C, the states $\mathbf{x} = (x - r, \dot{x}, \phi, \omega)$ and the input u are weighted by positive semi-definite matrices $\mathbf{Q}$ and R. The time horizon is 5 s ($N = 350$). Here we have defined performance as the cost improvement relative to 95% of the initial controller cost. The safe threshold is set at $J_{\min} = 0$. In practice, $J_{\min}$ can be chosen freely; however, we cannot set the threshold equal to the performance of the initial controller, as this does not allow the algorithm to classify nearby states as safe and expand the safe set.

While the optimal controller gains could be easily computed given an accurate model of the system, we do not have a model of the dynamics of the proprietary, on-board controller and the time delays in the system. Moreover, we want to optimize the performance for the real, nonlinear quadrotor system, which is difficult to model accurately. An inaccurate model of the system could be used to improve the prior GP model of the performance function, with the goal of achieving faster convergence. In this case, the uncertainty in the GP model of the performance function would account for inaccuracies in the system model.

We discretize the controller parameter space uniformly into $10,000$ combinations in $[-0.6, 0.1]^2$, explicitly including positive controller parameters that certainly lead to crashes. In practice, one would exclude parameters that are known to be unsafe *a priori*. The initial controller gains are $(-0.4, -0.4)$, which result in a controller with poor performance. Decreasing the controller gains further leads to unstable controllers.

To run the optimization algorithm we need to define a kernel for the GP. In this work, we choose the Matèrn kernel with parameter $\nu = 3/2$ [8],

$$k(\mathbf{a}_i, \mathbf{a}_j) = \sigma_\eta^2 \left(1 + \sqrt{3}\, r(\mathbf{a}_i, \mathbf{a}_j)\right) \exp\left(-\sqrt{3}\, r(\mathbf{a}_i, \mathbf{a}_j)\right), \tag{17}$$

$$r(\mathbf{a}_i, \mathbf{a}_j) = \sqrt{(\mathbf{a}_i - \mathbf{a}_j)^{\mathrm{T}}\mathbf{M}^{-2}(\mathbf{a}_i - \mathbf{a}_j)}, \tag{18}$$

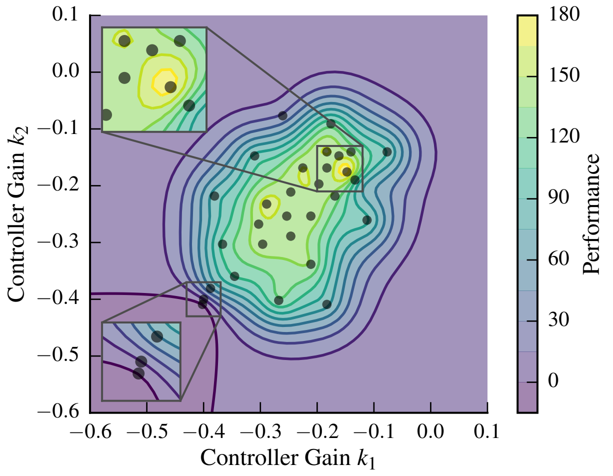


Fig. 3. GP mean estimate of the performance function after 30 evaluations. The algorithm adaptively decides where to sample based on safety and informativeness. In the bottom-left corner, there is the magnified section of the first three samples, which are close together to determine the location of the initial, safe region. The maximum, magnified in the top-left corner, also has more samples to determine the precise location of the maximum. Other areas are more coarsely sampled to expand the safe region.

which is parameterized by three hyperparameters: measurement noise, σ_ω^2 in (2) and (3), prior variance, σ_η^2, and postitive length-scales, $\mathbf{l} \in \mathbb{R}_+^{|\mathcal{A}|}$, which are the diagonal elements of the diagonal matrix $\mathbf{M}$, $\mathbf{M} = \mathrm{diag}(\mathbf{l})$, and correspond to the rate of change of the function J with respect to $\mathbf{a}$. This kernel function implies that the underlying function J is differentiable, takes values within the 2σ confidence interval $[-2\sigma_\eta, 2\sigma_\eta]$ with high probability and, with high probability, has a Lipschitz constant that depends on $\mathbf{l}$ and σ_η^2. These hyperparameters encode our prior assumptions about the unknown performance function. While it may be difficult to find hyperparameters that describe the underlying function perfectly, it is usually possible to specify parameters that are conservative (i.e., large σ_ω^2 and σ_η^2, and small length-scales, $\mathbf{l}$). In general, the more assumptions can be made (e.g., smoothness), the faster the algorithm will converge. This leads to a trade-off between ensuring that the function is well modeled (that is, the safe threshold is not violated), and the number of experiments we are willing to conduct.

The parameters for the experiments were set as follows: the length-scales were set to 0.05 for both parameters, which corresponds to the notion that a 0.05-0.1 change in the parameters leads to very different performance values. The prior standard deviation, σ_η, and the noise standard deviation, σ_ω, are set to 5% and 10% of the performance of the inital controller, $C(\mathbf{a}_0)$, respectively. The noise standard deviation, σ_ω, mostly models errors due to initial position offsets, since state measurements have low noise. The size of these errors depends on the choice of the matrices $\mathbf{Q}$ and R. By choosing σ_ω dependent on the initial performance, we account for the $\mathbf{Q}$ and R dependency. Similarly, σ_η specifies the expected size of the performance function values. Initially, the best we can do is to set this quantity dependent on the initial performance and leave additional room for future, larger performance values.

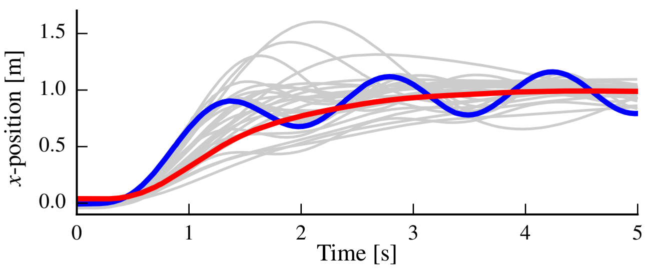


Fig. 4. The quadrotor controller performance is evaluated during a 5 s evaluation interval, where a 1 m reference position change must be performed. The trajectories correspond to the optimization routine in Fig. 3. The initial controller (blue) performs poorly but is stable. In contrast, the optimized controller (red) shows an optimized, smooth, and fast response. The trajectories of other controller parameters that were evaluated are shown in gray.

The resulting, estimated performance function after running Algorithm 1 for 30 experiments is shown in Fig. 3. The unknown function has been reliably identified. Samples are spread out over the entire safe set, with more samples close to the maximum of the function and close to the initial controller parameters. No unsafe parameters below the threshold $J_{\min} = 0$ were evaluated on the real system.

Typically, the optimization behavior of Algorithm 1 can be roughly separated into three stages. Initially, the algorithm evaluates controller parameters close to the initial parameters in order for the GP to acquire information about the safe set (see lower-left, zoomed-in section in Fig. 3). Once a region of safe controller parameters is determined, the algorithm evaluates the performance function more coarsely in order to expand the safe set. Eventually, the controller parameters are refined by evaluating high-performance parameters that are potential maximizers in a finer grid (see upper-left, zoomed-in section in Fig. 3). The trajectories of the initial, best and intermediate controllers can be seen in Fig. 4.

A normal Bayesian optimization algorithm [7] would start by evaluating $\mathbf{a} = (0.1,\ 0.1)$, where the GP has the largest uncertainty about the performance. These parameters lead to an unstable controller. In contrast, our method safely explores the parameter space without evaluating unsafe parameters.

Ultimately, the algorithm identifies the controller gains that maximize the performance measure. Because we omitted the on-board controller and its internal states and due to the nonlinearity of the quadrotor dynamics, the resulting performance function in Fig. 3 is similar to, but not the same as, the quadratic function that one would have expected from linear quadratic control theory.

VI. CONCLUSION

We presented the first application of Safe Bayesian Optimization on a real robotic system. We modified the algorithm in [14] to work directly with GP estimates and successfully applied it to the position control of a quadrotor vehicle. It was shown that the algorithm enables efficient, automatic, and global optimization of the controller parameters without risking dangerous and expensive system failures.

REFERENCES

[1] F. Berkenkamp, A. P. Schoellig, and A. Krause, "SafeOpt source code," *GitHub*, 2016, http://github.com/befelix/SafeOpt-robotics.

[2] N. J. Killingsworth and M. Krstić, "PID tuning using extremum seeking: online, model-free performance optimization," *IEEE Control Systems*, vol. 26, no. 1, pp. 70–79, 2006.

[3] K. J. Åström, T. Hägglund, C. C. Hang, and W. K. Ho, "Automatic tuning and adaptation for PID controllers - a survey," *Control Engineering Practice*, vol. 1, no. 4, pp. 699–714, 1993.

[4] Y. Davidor, *Genetic algorithms and robotics: a heuristic strategy for optimization*. World Scientific, 1991.

[5] J. Mockus, *Bayesian approach to global optimization: theory and applications*. Springer Science & Business Media, 2012.

[6] A. D. Bull, "Convergence rates of efficient global optimization algorithms," *Journal of Machine Learning Research*, vol. 12, pp. 2879–2904, 2011.

[7] N. Srinivas, A. Krause, S. M. Kakade, and M. Seeger, "Gaussian process optimization in the bandit setting: no regret and experimental design," in *Proc. of the International Conference on Machine Learning (ICML)*, 2010, pp. 1015–1022.

[8] C. E. Rasmussen and C. K. Williams, *Gaussian processes for machine learning*. MIT Press, 2006.

[9] D. R. Jones, "A taxonomy of global optimization methods based on response surfaces," *Journal of Global Optimization*, vol. 21, no. 4, pp. 345–383, 2001.

[10] R. Calandra, N. Gopalan, A. Seyfarth, J. Peters, and M. P. Deisenroth, "Bayesian gait optimization for bipedal locomotion," in *Learning and Intelligent Optimization*. Springer, 2014, pp. 274–290.

[11] D. J. Lizotte, T. Wang, M. H. Bowling, and D. Schuurmans, "Automatic gait optimization with Gaussian process regression." in *Proc. of the International Joint Conference on Artificial Intelligence (IJCAI)*, vol. 7, 2007, pp. 944–949.

[12] M. Tesch, J. Schneider, and H. Choset, "Using response surfaces and expected improvement to optimize snake robot gait parameters," in *Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2011, pp. 1069–1074.

[13] A. Marco, P. Hennig, J. Bohg, S. Schaal, and S. Trimpe, "Automatic LQR tuning based on Gaussian process global optimization," in *Proc. of the IEEE International Conference on Robotics and Automation (ICRA)*, 2016.

[14] Y. Sui, A. Gotovos, J. W. Burdick, and A. Krause, "Safe exploration for optimization with Gaussian processes," in *Proc. of the International Conference on Machine Learning (ICML)*, 2015, pp. 997–1005.

[15] F. Berkenkamp, A. P. Schoellig, and A. Krause, "Safe and automatic controller tuning with Gaussian processes," in *Proc. of the Workshop on Machine Learning in Planning and Control of Robot Motion, IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2015.

[16] A. Aswani, H. Gonzalez, S. S. Sastry, and C. Tomlin, "Provably safe and robust learning-based model predictive control," *Automatica*, vol. 49, no. 5, pp. 1216–1226, 2013.

[17] A. K. Akametalu, S. Kaynama, J. F. Fisac, M. N. Zeilinger, J. H. Gillula, and C. J. Tomlin, "Reachability-based safe learning with Gaussian processes," in *Proc. of the IEEE Conference on Decision and Control (CDC)*, 2014, pp. 1424–1431.

[18] T. M. Moldovan and P. Abbeel, "Safe exploration in Markov decision processes," in *Proc. of the International Conference on Machine Learning (ICML)*, 2012, pp. 1711–1718.

[19] M. A. Gelbart, J. Snoek, and R. P. Adams, "Bayesian optimization with unknown constraints," in *Proc. of the Conference on Uncertainty in Artificial Intelligence (UAI)*, 2014, pp. 250–259.

[20] J. Schreiter, D. Nguyen-Tuong, M. Eberts, B. Bischoff, H. Markert, and M. Toussaint, "Safe exploration for active learning with Gaussian processes," in *Proc. of the European Conference on Machine Learning (ECML)*, vol. 9284, 2015, pp. 133–149.

[21] F. Berkenkamp and A. P. Schoellig, "Safe and robust learning control with Gaussian processes," in *Proc. of the European Control Conference (ECC)*, 2015, pp. 2501–2506.

[22] F. Berkenkamp, A. Krause, and Angela P. Schoellig, "Bayesian optimization with safety constraints: safe and automatic parameter tuning in robotics." arXiv, 2016, arXiv:1602.04450 [cs.RO].